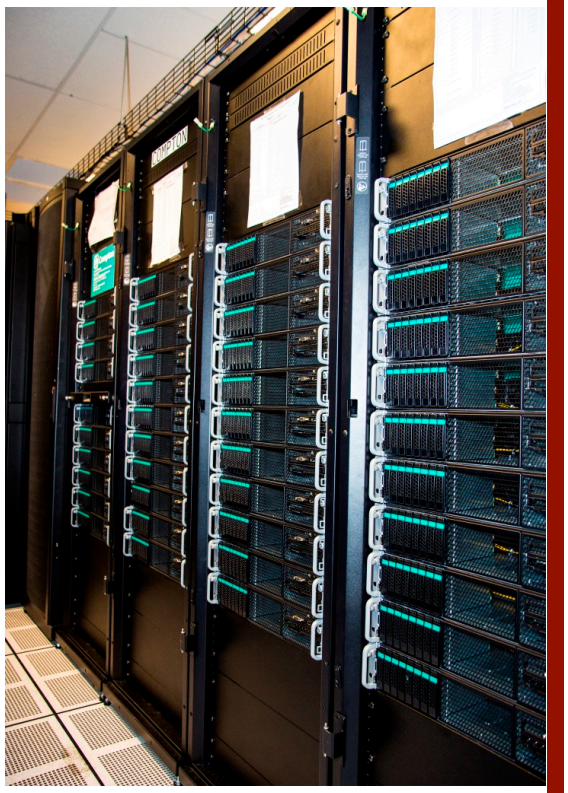
CCR
Center for Computing Research

Kokkos
Hierarchical Task-Data Parallelism for C++ HPC Applications

GPU Tech. Conference
April 4-7, 2016
San Jose, CA

H. Carter Edwards

SAND2016-3114 C



Sandia National Laboratories

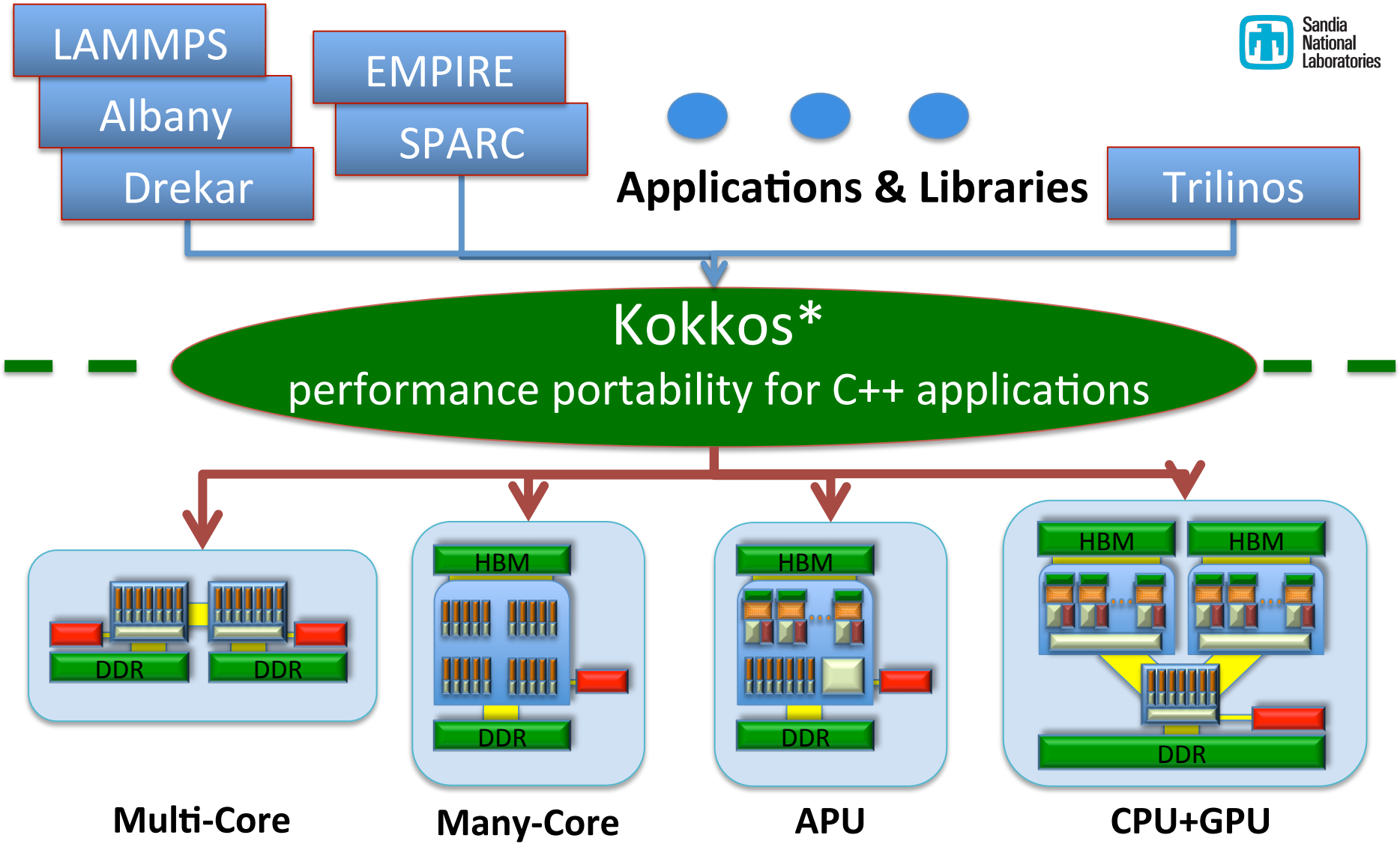
Exceptional service in the national interest

Advanced Simulation & Computing

U.S. Department of Energy

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP



*KÓKKOS Greek: "granule" or "grain" ; like grains of sand on a beach

Abstractions: Patterns, Policies, and Spaces

Sandia National Laboratories

- **Parallel Pattern** of user's computations
  - parallel_for, parallel_reduce, parallel_scan, ... ***EXTENSIBLE***
- **Execution Policy** tells ***how*** user computation will be executed
  - Static scheduling, dynamic scheduling, thread-teams, ... ***EXTENSIBLE***
- **Execution Space** tells ***where*** user computations will execute
  - Which cores, numa region, GPU, ... *(extensible)*
- **Memory Space** tells ***where*** user data resides
  - Host memory, GPU memory, high bandwidth memory, ... *(extensible)*
- **Array Layout** (policy) tells ***how*** user data is laid out in memory
  - Row-major, column-major, array-of-struct, struct-of-array ... *(extensible)*
- Differentiating feature: Array Layout and Memory Space
  - Versus other programming models (OpenMP, OpenACC, ...)
  - Critical for performance portability ... see previous GPU-Tech Kokkos talks

Acknowledgements

- **Sandia National Laboratory (SNL)**

  Laboratory Directed Research & Development (LDRD) Project
  - **Strategic R&D funded at Sandia's discretion**
  - **Recognized with $$ as strategic high priority**
- **SNL LDRD Team**
  - **myself, Stephen Olivier, Jonathan Berry, Greg Mackey, Siva Rajamanickam, Kyungjoo Kim, George Stelle, and Michael Wolf**
  - **Scheduling algorithm inspired from SNL's Qthreads library**
- **Critical support from NVIDIA**
  - **Thanks to Tony Scudiero, Greg Branch, Ujval Kapasi, and the whole nvcc development team**
  - **Early access to nvcc CUDA 8 essential for relocatable device code**

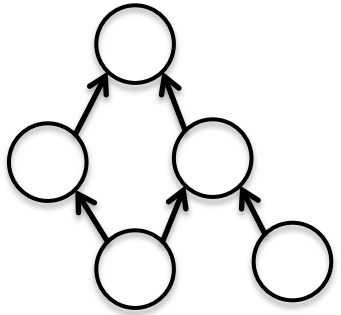
Sandia National Laboratories

New (prototype) Kokkos Capability:

Dynamic Directed Acyclic Graph (DAG) of Tasks

- **Extension of Parallel Pattern**
  - **Tasks: Heterogeneous collection of parallel computations**
  - **DAG: Tasks may have acyclic "execute after" dependences**
  - **Dynamic: New tasks may be created/allocated by executing tasks**

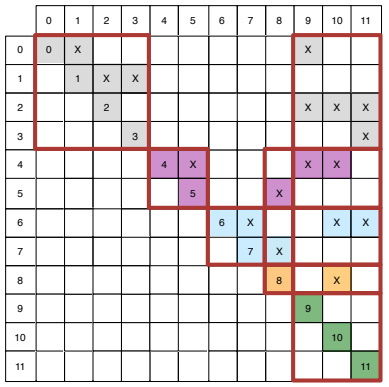
- **Extension of Execution Policy**
  - **Schedule tasks for execution**
  - **Manage tasks' dynamic lifecycle**

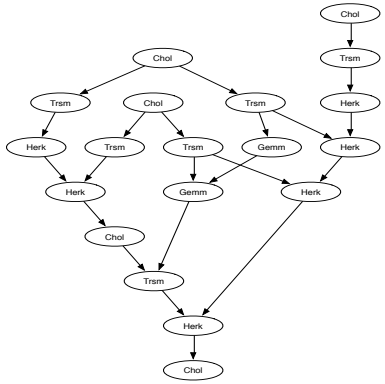


Use Cases (mini-applications)

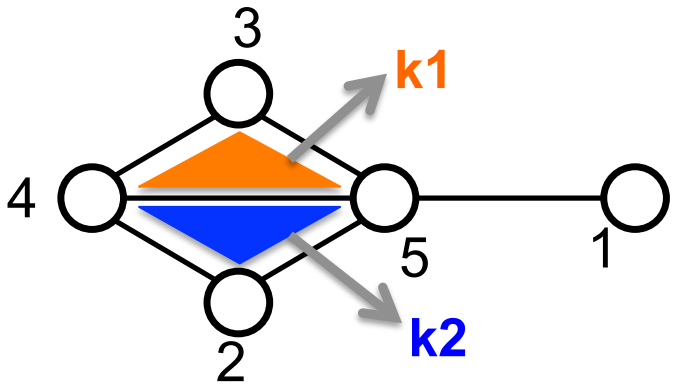
Sandia National Laboratories

1. **Incomplete Level-K Cholesky factorization of sparse matrix**
   - Block partitioning into submatrices
   - DAG of submatrix computations
   - Each submatrix computation is internally data parallel
   - Lead: Kyungjoo Kim





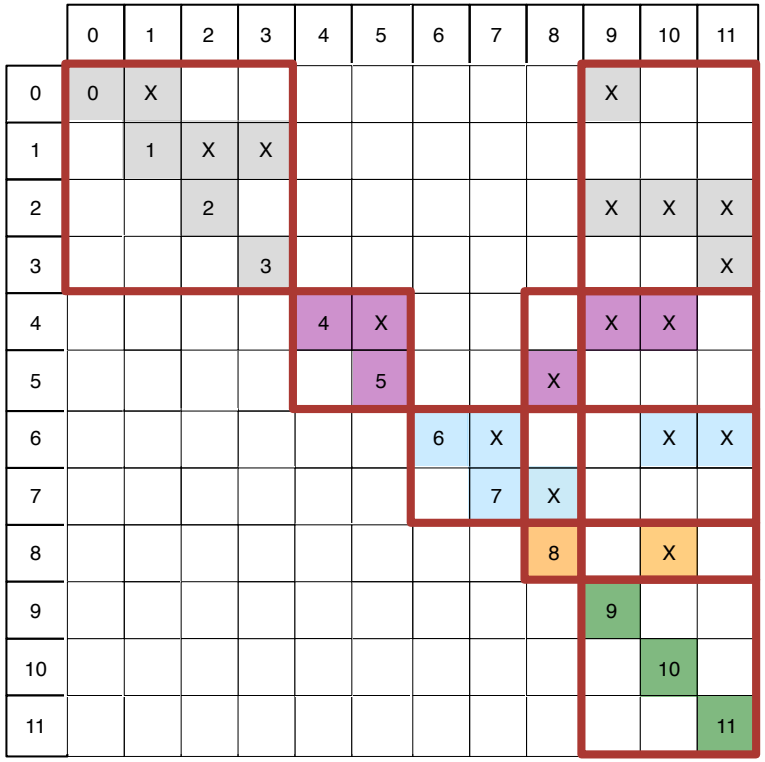
2. **Triangle enumeration in social networks (highly irregular graphs)**
   - Identify triangles within the graph
   - Compute statistics on triangles
   - Triangles are an intermediate result that do not need to be saved / stored
     - Problem: memory "high water mark"
   - Lead: Michael Wolf

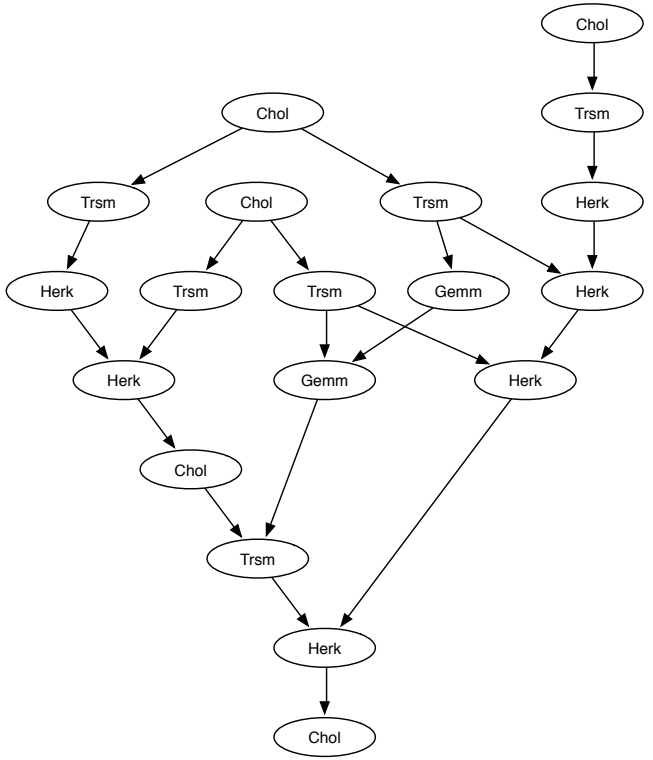


Use Case 1: Incomplete Cholesky Factorization

Sandia National Laboratories

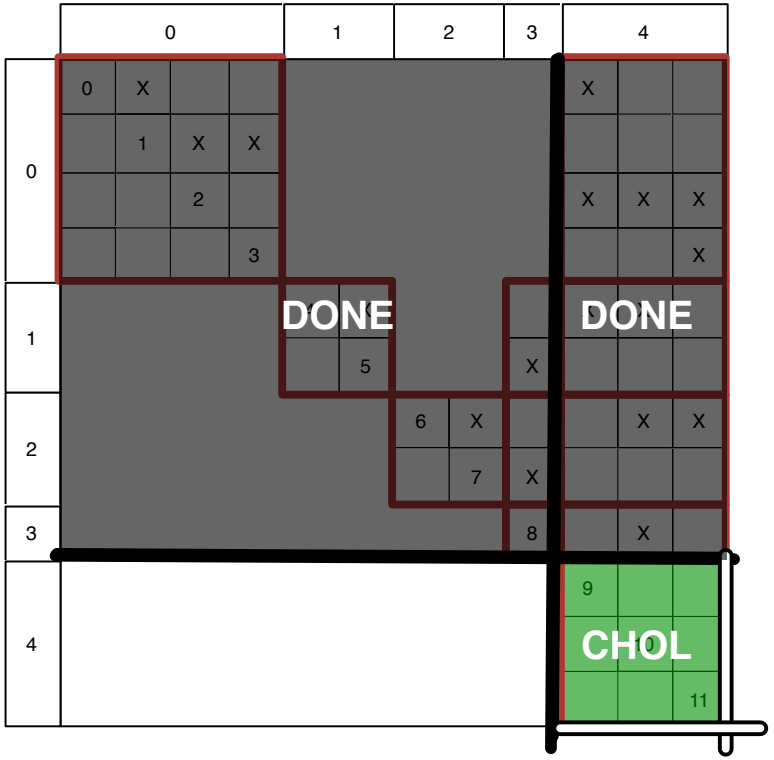
- Reordering and Block Partitioning of Sparse Matrix
- One driver task that creates and spawns submatrix task-DAG
- Submatrix tasks factor, triangular solve, rank-k update, multiply

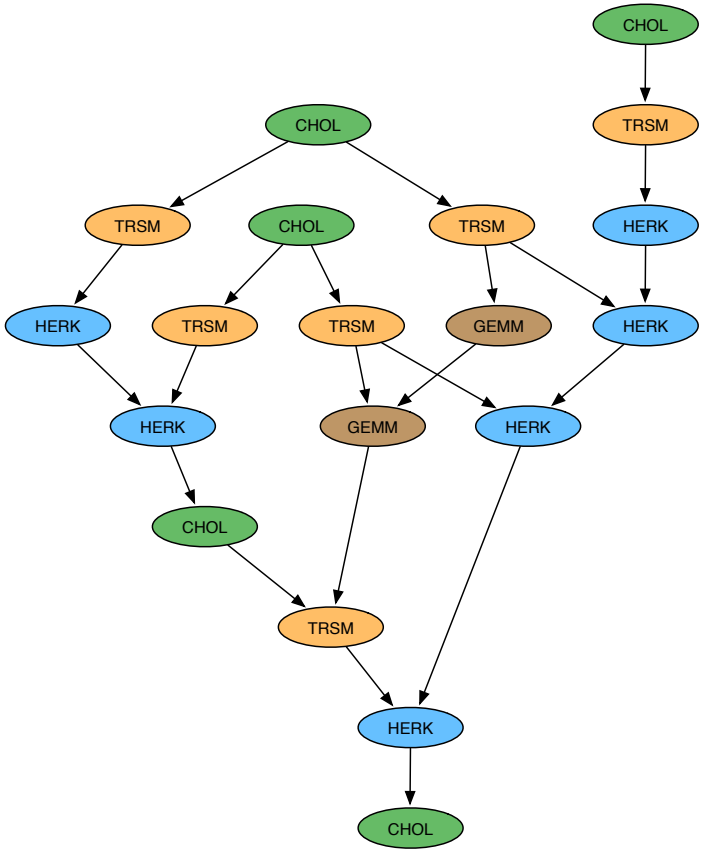




Use Case 1: Incomplete Cholesky Factorization Driver Task: Spawn Submatrix Task-DAG

Sandia National Laboratories





- Periodically stop iterating and respawn this task with low priority
  - Throttle back submatrix task generation for memory constraint
  - Allow submatrix tasks to complete and their memory to be reclaimed

Use Case 2: Triangle Enumeration and Statistics of Social Network

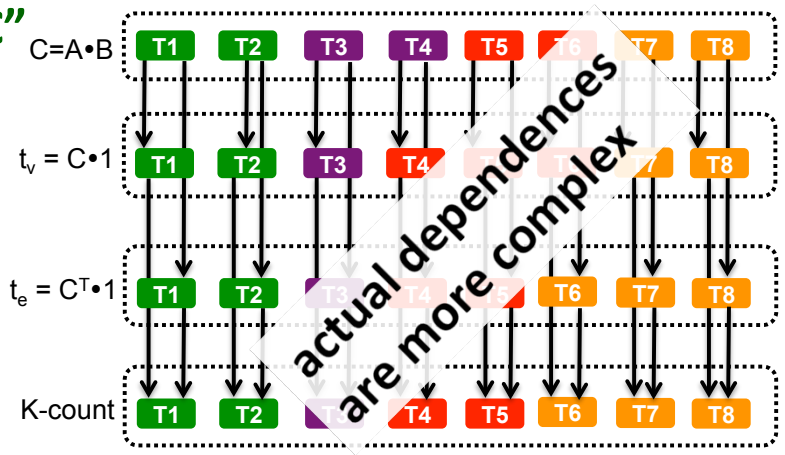
Sandia National Laboratories

- **miniTri: proxy (mini-application) for triangle based data analytics**
- **Current linear-algebra strategy**
  - **"A" is the adjacency matrix**
  - **"B" is the incidence matrix**

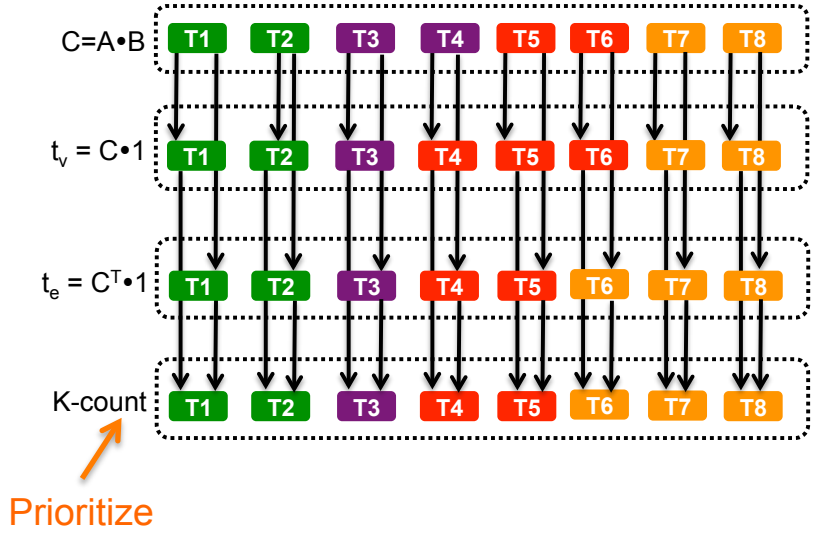
miniTri

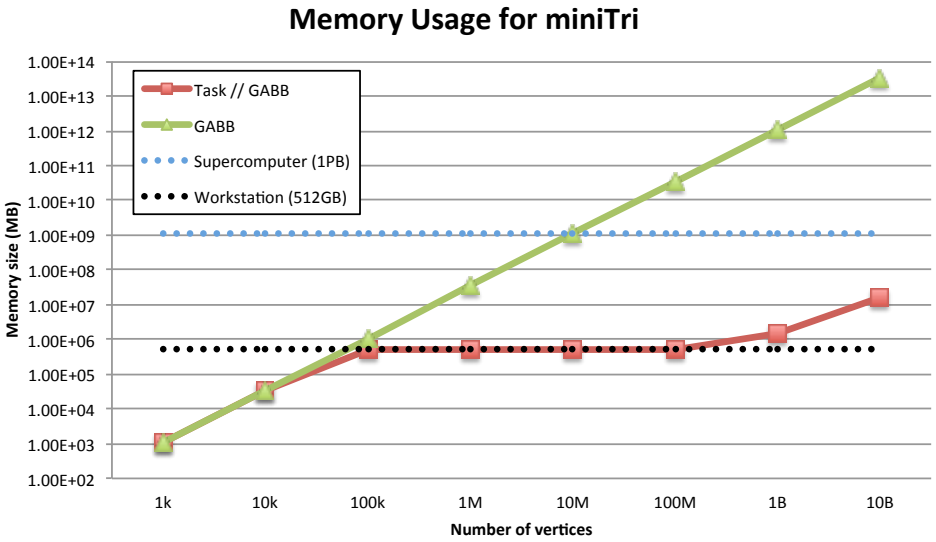
1. $C = A * B$
2. $t_v = C * 1$
3. $t_e = C^T * 1$
4. $kcount(C, t_v, t_e)$

- **Challenges**
  - **Very irregular graph, difficult to statically load balance**
  - **Graph BLAS strategy explicitly forms "C" which is all triangles in the graph**
  - ➢ **Extremely large intermediate storage of "C"**
- **Task parallelism pipelines operations**
  - **Each phase is a data parallel BLAS**
  - **Block partition and pipeline via DAG**
  - **Prioritize downstream tasks to "retire" temporary "C" submatrices**



Task Parallelism for Resource Constraints





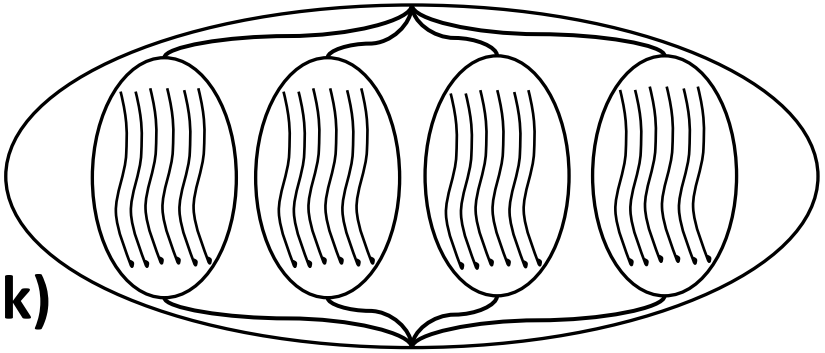
- Key insight: Task parallelism can be used to reduce memory footprint
  - Prioritize k-count tasks to free blocks of triangles from memory
  - Need runtime system to support advanced resource management/ priorities (ongoing effort: HPX and Kokkos/Qthreads)

Task parallel approach allows Graph BLAS implementation of miniTri to solve much larger problems

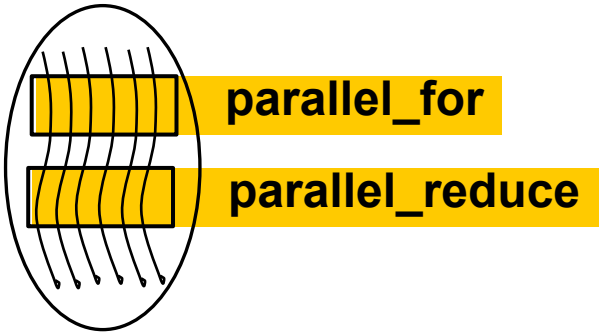
Hierarchical Parallelism

Sandia National Laboratories

- **Shared functionality with hierarchical data-data parallelism**
  - The same kernel (task) executed on …
  - OpenMP: League of Teams of Threads
  - Cuda: Grid of Blocks of Threads



- **Intra-Team Parallelism (data or task)**
  - Threads within a team execute concurrently
  - Data: each team executes the same computation
  - Task: each team executes a different task
  - ➢ **Nested parallel patterns: for, reduce, scan**

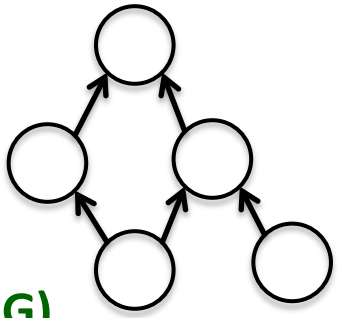


- **Mapping teams onto hardware**
  - CPU : team == hyperthreads sharing L1 cache
    - Requires low degree of intra-team parallelism
  - Cuda : team == thread block
    - Requires high degree of intra-team parallelism
    - *… revisit this later*

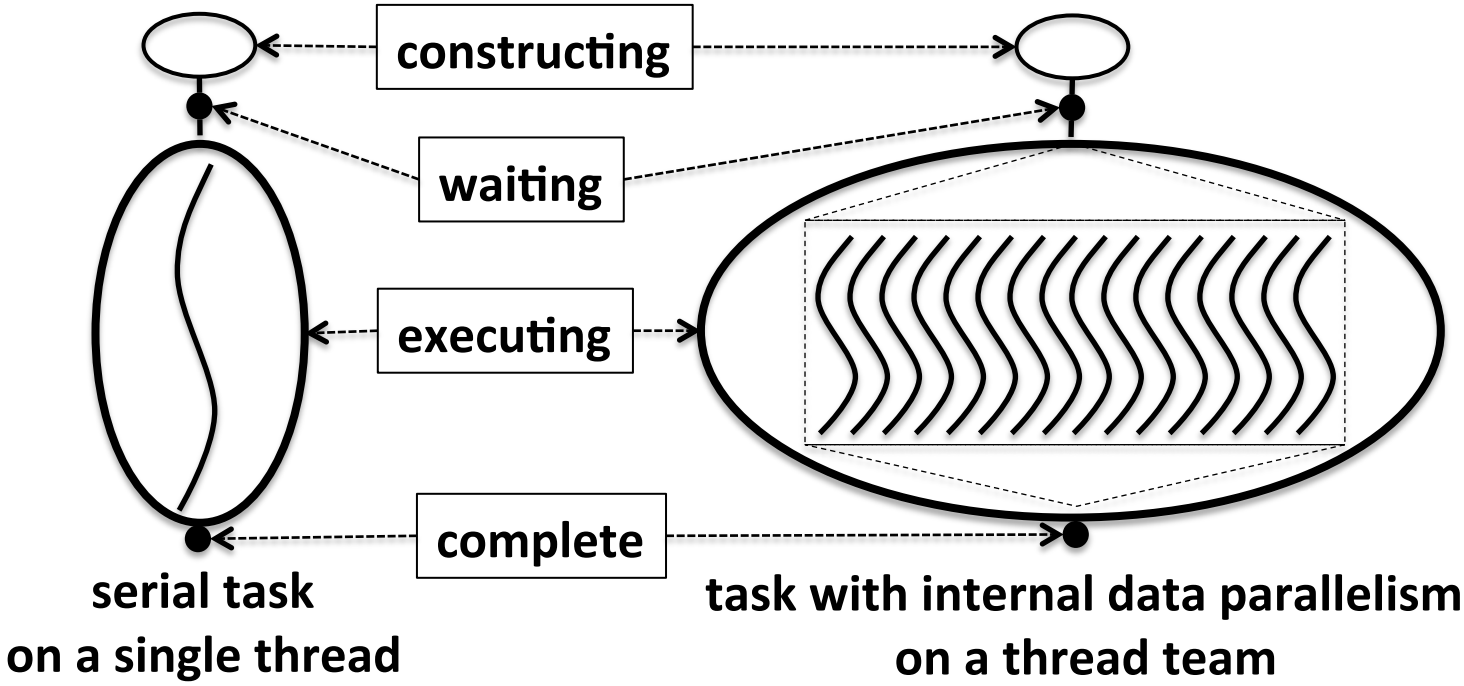
Anatomy and Life-cycle of a Task

Sandia National Laboratories

- **Anatomy**
  - Is a C++ closure (e.g., functor) of data + function
  - Is referenced by a *Kokkos::future*
  - Executes on a single thread or thread team
  - May only execute when its dependences are complete (DAG)



- **Life-cycle:**



Dynamic Task DAG Execution Policy

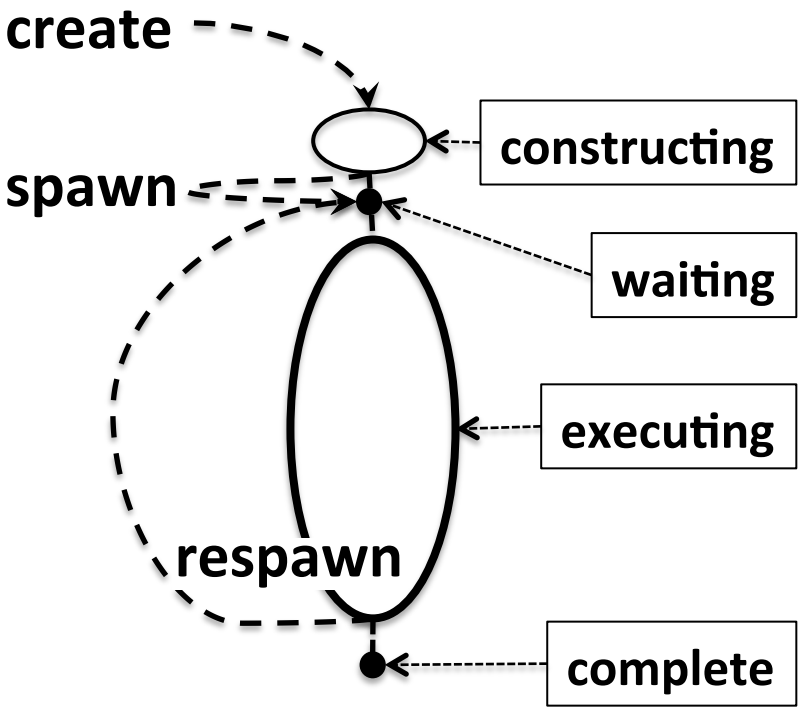
Sandia National Laboratories

- Manage a *heterogeneous* collection of tasks
  - Map task execution to a thread team or single thread
  - Execution constrained by task dependence DAG
  - Memory management for tasks' *dynamically* allocated memory
- Challenges
  - Portability across multicore/manycore architectures: CPU, GPU, Xeon Phi, …
    - GPU function pointer accessibility on host and device
  - Dynamic – creating tasks within executing tasks on GPU
  - Performance – thread scalable allocation/deallocation within finite memory
  - Performance – execution overhead and thread scalable scheduling
- Non-blocking constraint for portability and performance
  - An executing task ***cannot*** block or yield
  - Eliminates overhead of saving execution state: registers, stack, …
  - Reduces overhead of context switching

Managing a Non-blocking Task's Lifecycle

Sandia National Laboratories

- Create: allocate and construct
  - By main process or within another task
  - Allocate from a memory pool
  - Construct internal data
  - Assign DAG dependences
- Spawn: enqueue to scheduler
- Respawn: re-enqueue to scheduler
  - Instead of the task waiting or yielding
  - Can reassign DAG dependences
    - Essential capability for mini-Tri
  - Used by Cholesky factorization to throttle back task generation
  - Reconceived wait-for-child-task use case
    - Create & spawn child task(s)
    - Reassign DAG dependence(s) to new child task(s)
    - Re-spawn to execute again after child task(s) complete



CPU/GPU Portable Scheduler

Sandia National Laboratories

- **Multiple Queues of Tasks**
  - Waiting on incomplete task(s); i.e. task dependences
  - Multiple ready to execute queues with priorities: high, *regular*, low
  - Queues managed with atomic operations
- **Persistent Threads Strategy (CPU pthreads, GPU thread blocks)**
  - Pop ready tasks, by priority, and execute
  - When task is complete update dependent tasks; e.g., move to ready queue
  - When task is dereferenced (last future is destroyed) reclaim memory
- **Constraint: Tasks reside within fixed size Memory Pool**
  - Tasks can create (allocate) and spawn new tasks, even on GPU
  - Algorithms must account for fixed size constraint when creating new tasks
    - E.g., Incomplete Cholesky factorization throttles back task creation

Use Case 1: Incomplete Cholesky Factorization GPU unacceptable Evaluation

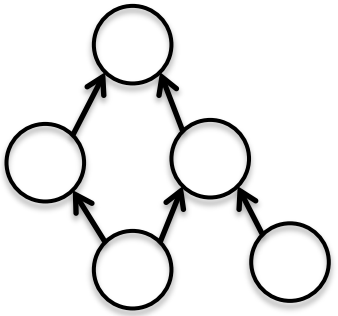
Sandia National Laboratories

- **Initial prototype, to-be-done improvements & optimization**
  - ✓ **Successfully executes dynamic task-DAG**
- **Recall thread-team tasks are mapped to CUDA thread blocks**
  - ➢ **Requires high degree of intra-team parallelism**
- **Sparse submatrix tasks use intra-team parallelism**
  - **Dominated by non-coalesced indirect memory access (reference chasing)**
  - **Insufficient work, low computational intensity, high register usage**
  - ➢ **12% occupancy *and* poor memory access patterns**
- **To-do Improvement: map thread-team tasks to CUDA warps**
  - **Requires refactoring of intra-team parallel patterns and policies**
  - **The revisit memory access patterns**
  - ***… stay tuned***

Conclusion and Ongoing R&D

Sandia National Laboratories

- ✓ **Prototype Portable Dynamic Task-DAG**
  - Portable: CPU and NVIDIA GPU architectures
  - Collection of heterogeneous parallel computations; a.k.a., tasks
  - With directed acyclic graph (DAG) of task dependences
  - Dynamic – tasks may create tasks
  - Hierarchical – thread-team data parallelism within tasks



- **Challenges, primarily for GPU portability and performance**
  - Non-blocking tasks ➔ respawn instead of wait
  - Memory pool for dynamically allocatable tasks
  - TBD: Thread-team map onto GPU warps, not thread blocks
- **In progress: Refactoring of thread-team mapping for GPU**
  - ➢ Unfortunately, the clock ran out on us for GPU-Tech 2016
  - *… stay tuned*